RECEIVED
DEC 11 2002
Technology Center 2100

**IN THE SPECIFICATION:**

Please replace the title with the following:

OPERAND QUEUE FOR USE IN A FLOATING POINT UNIT TO REDUCE READ-AFTER-WRITE LATENCY AND METHOD OF OPERATION

Please replace the paragraphs bridging page 2, line 22 through page 3, line 20 with the following:

---

A\

Efficiency is particularly important in mathematical calculations, particularly floating point calculations. Some mathematical operations, such as multiplication and division, cause significant delays during program execution. A pipelined floating point unit (FPU) may be particularly susceptible to long delays during the execution of certain sequences of instructions. For example, a floating point "load" instruction may occur in a pipelined FPU immediately after, or shortly after, a floating point store instruction occurs. This is sometimes referred to as a "read-after-write" (RAW) hazard. The write (or store) operation to system memory may have a long latency before the write data is "committed" to system memory by the processor. The read (or load) operation following the write (or store) operation may occur before the

A1

write operation is complete and may, therefore, suffer significant delays waiting for the write operation to complete before the committed data may be read back from memory.

Therefore, there is a need in the art for an improved microprocessor that executes mathematical operations more rapidly. In particular, there is a need for an improved floating point unit that executes floating point operations as rapidly as possible. More particularly, there is a need in the art for a floating point unit that minimizes delays caused by writing data to memory.

Please replace the paragraph on page 5 at lines 16–21 of the specification with the following:

A2

In a further embodiment of the present invention, the data in the external memory is accessed in groups of N bytes and the floating point unit further comprises at least one aligner capable of receiving a first incoming operand that is misaligned with respect to a boundary between a first N byte group and a second N byte group and aligning the first incoming operand.

Please replace the paragraphs bridging page 9, line 10 through page 11, line 20 with the following:

A3

FIGURE 1 illustrates processing system 10, which includes integrated microprocessor 100, according to one embodiment of the present invention. Integrated microprocessor 100 comprises central processing unit (CPU) 105, which has dual integer and dual floating point execution units, separate load/store and branch units, and level one (L1) instruction and data caches. Microprocessor 100 also comprises graphics unit 110, system memory controller 115, and level two (L2) cache 120, which is shared by CPU 105 and graphics unit 110. Graphics unit 110, system memory controller 115, and L2 cache 120 may be integrated onto the same die as CPU 105. Bus interface unit 125 couples CPU 105, graphics unit 110, and L2 cache 120 to memory controller 115. Bus interface unit 125 also may be integrated onto the same die as CPU 105.

Integrated memory controller 115 bridges microprocessor 100 to system memory 140, and may provide data compression and/or decompression to reduce bus traffic over external memory bus 145 which preferably, although not exclusively, has a RAMbus™, fast synchronous dynamic random access memory (SDRAM) or other type protocol. Integrated graphics unit 110 provides thin film transistor (TFT), DSTN, red-green-blue (RGB), and other types of video output to drive display 150.

Bus interface unit 125 connects microprocessor 100 through input/output (I/O) interface 130 to PCI bridge 155, which has a conventional peripheral component interconnect (PCI) bus interface on PCI bus 160 to one or more

A3

peripherals, such as sound card 162, local area network (LAN) controller 164, and disk drive 166, among others. Bus interface unit 125 also connects fast serial link 180 and relatively slow I/O port 185 to microprocessor 100 (via I/O interface 130 and PCI bridge 155). Fast serial link 180 may be, for example, an IEEE 1394 bus (i.e., "Firewire") and/or a universal serial bus ("USB"). I/O port 185 is used to connect peripherals to microprocessor 100, such as keyboard 190 and/or a mouse. In some embodiments, PCI bridge 155 may integrate local bus functions such as sound, disk drive control, modem, network adapter, and the like.

---

Please replace the paragraph on page 12 at lines 8–18 of the specification with the following:

---

A4

In the exemplary embodiment, FPU 230 uses two load buses because the frequency of load operations is twice the frequency of floating point operations. Therefore, in order to achieve an execution rate of one floating point operation per clock, FPU 230 uses two load buses 240. FPU 230 uses one store bus 245 to store results to system memory 140 at commit time. Unlike load operations, where the memory alignment is done in FPU 230, rotating data to put it in memory format is done in data cache 220. The reason for one store bus is that store operations only comprise between 5% and 15% of all floating point instructions, so one bus is sufficient for bandwidth purposes.

---

Please replace the paragraph bridging page 13, line 14 through page 14, line 5 of the specification with the following:

A5

FPU 230 receives opcodes (instructions) from instruction decoder/Ucode logic 210. Since the number of bits required to control FPU 230 may be quite large, instruction decoder/Ucode logic 210 does not send FPU 230 a micro-word. Instead, instruction decoder/Ucode logic 210 sends index values to FPU micro-ROM (UROM) 302. The index values are represented by the inputs instruction/microcode (IU) index (0) to instruction/microcode (IU) index (3). UROM 302 outputs consists of an add/multiply operation and a load store operation that are applied to node exchange (XCH)/register mapping logic and logical-to-physical register file logic 304. XCH/Reg & Mapping and LRF logic 304 computes the physical source and destination addresses in system memory 140 of an operand for each instruction in system memory 140 using register offset values represented by inputs register offset (0) through register offset (3).

Please replace the paragraphs bridging page 14, line 18 through page 15, line 15 of the specification with the following:

A6

Finally, the data may not have been computed yet. In this final case, the

A6

dependent instruction is marked as pending and the PRF location where the data will be deposited is returned. The dependent instruction then monitors the result busses and when the result is produced, PRF 330 is read to obtain the data. Once the operation and physical locations of the operands have been generated, the opcodes are loaded into opcode queues 341-344 associated with each functional unit and into a content addressable memory (CAM) which controls the operand valid bits.

There are four major functional units in FPU 230. Adder 311 and multiplier 313 perform the majority of the arithmetic. These operations are fully pipelined and have a latency of three clock cycles and a throughput of one clock cycle. FPU 230 uses two load conversion units 315a and 315b to convert load data from a format stored in system memory 140 to the internal format of FPU 230. Load conversion units 315a and 315b receive operands only from operand queue 345. When all pieces of load data in operand queue 345 are valid, one of load conversion units 315a and 315b is scheduled to convert the load data. The opcode in opcode queue 343 indicates how wide the load data is and what format conversion the load data requires.

---

Please replace the paragraph on page 19 at lines 3–11 of the specification with the following:

---

A7

As used herein, "virtual commit" is the process of transferring store data from

A7

operand queue 345 into virtual commit buffer 350, as well as storing virtually committed data into any dependent load slots in operand queue 345. The process of virtual commit is performed on a slot-by-slot basis in operand queue 345 and virtual commit buffer 350. However, a virtual commit cycle is only required if a slot has a floating-point store in it. Checkpoints that do not have any floating-point stores also require 1 cycle to virtually commit.

Please replace the paragraphs on page 20 at lines 3–21 of the specification with the following:

A8

The virtual commit pointer is advanced as quickly as it can be through the slots in virtual commit buffer 350. This means that the virtual commit pointer does not wait for a store to complete for it to advance. Instead, as soon as a checkpoint has been issued to the load/store unit, the virtual commit pointer pulls all stores from operand queue 345 and forwards data from the store operation to any dependent read operation. The virtual commit pointer only stops after all stores for the three virtual commit checkpoints have been read.

When a store occurs, the store data is written into operand queue 345 at the address indexed by the store slot:checkpoint value and the CAMs in forwarding array 351 compare the store address with all “forward from” addresses so that all

A8

dependent reads will be updated as well. The CAM outputs are used as word lines for operand queue 345 and are also used to mark the dependent reads as needing re-execution. Store operations also write into virtual commit buffer 350 at the proper slot:checkpoint value, so that it is not necessary to back up the virtual commit pointer to the slot:checkpoint value where the store occurred.

---

Please replace the paragraph on page 32, at lines 5–18 of the specification (the "ABSTRACT OF THE DISCLOSURE") with the following:

---

A9

A floating point unit includes floating point processing units for executing floating point instructions that write operands to an external memory and for executing floating point instructions that read operands from the external memory. The floating point also includes an operand queue for storing a plurality of operands associated with one or more operations being processed in the floating point unit. The operand queue stores a first operand written by a floating point write instruction executed by a first one of the plurality of floating point processing units and supplies the first operand to a floating point read instruction executed by a second one of the plurality of floating point processing units when the first operand is committed or virtually committed.

---